**Complete Docker Commands Reference Guide**

**Images**

**Basic Image Commands**

# List all images

docker images

docker image ls

# Pull an image from registry

docker pull <image\_name>:<tag>

docker pull nginx:latest

# Build an image from Dockerfile

docker build -t <image\_name>:<tag> .

docker build -t myapp:1.0 /path/to/dockerfile

# Remove an image

docker rmi <image\_id>

docker rmi <image\_name>:<tag>

# Remove all unused images

docker image prune

# Show image history

docker history <image\_name>

# Inspect image details

docker inspect <image\_name>

# Tag an image

docker tag <source\_image> <target\_image>

# Push image to registry

docker push <image\_name>:<tag>

# Save image to tar file

docker save -o <filename>.tar <image\_name>

# Load image from tar file

docker load -i <filename>.tar

**Containers**

**Container Lifecycle**

# Run a container

docker run <image\_name>

docker run -d --name <container\_name> <image\_name>

# Run with port mapping

docker run -p <host\_port>:<container\_port> <image\_name>

# Run with environment variables

docker run -e ENV\_VAR=value <image\_name>

# Run with volume mount

docker run -v <host\_path>:<container\_path> <image\_name>

# Run interactively

docker run -it <image\_name> /bin/bash

# List running containers

docker ps

# List all containers (including stopped)

docker ps -a

# Stop a container

docker stop <container\_id>

# Start a stopped container

docker start <container\_id>

# Restart a container

docker restart <container\_id>

# Remove a container

docker rm <container\_id>

# Remove all stopped containers

docker container prune

# Execute command in running container

docker exec -it <container\_id> <command>

# View container logs

docker logs <container\_id>

docker logs -f <container\_id> # Follow logs

# Copy files between host and container

docker cp <container\_id>:<path> <host\_path>

docker cp <host\_path> <container\_id>:<path>

# Show container resource usage

docker stats <container\_id>

# Inspect container details

docker inspect <container\_id>

**Handling Codes & Objects**

**Development Workflow**

# Build and run application

docker build -t myapp .

docker run -d -p 8080:80 myapp

# Hot reload development

docker run -v $(pwd):/app -p 8080:80 myapp

# Multi-stage builds

docker build --target development .

docker build --target production .

# Build with build arguments

docker build --build-arg ARG\_NAME=value .

# Export container as image

docker commit <container\_id> <new\_image\_name>

# Create container without starting

docker create --name <container\_name> <image\_name>

# Pause/unpause container

docker pause <container\_id>

docker unpause <container\_id>

# Wait for container to exit

docker wait <container\_id>

# Show processes running in container

docker top <container\_id>

# Show container port mappings

docker port <container\_id>

**Data Management & Cache**

**Data Persistence**

# Create named volume

docker volume create <volume\_name>

# List volumes

docker volume ls

# Inspect volume

docker volume inspect <volume\_name>

# Remove volume

docker volume rm <volume\_name>

# Remove unused volumes

docker volume prune

# Bind mount (host directory)

docker run -v /host/path:/container/path <image>

# Named volume mount

docker run -v <volume\_name>:/container/path <image>

# Temporary filesystem (tmpfs)

docker run --tmpfs /tmp <image>

# Show disk usage

docker system df

# Clean up everything

docker system prune

# Clean up with volumes

docker system prune --volumes

# Remove build cache

docker builder prune

**Dockerfiles**

**Dockerfile Instructions**

# Base image

FROM ubuntu:20.04

# Set working directory

WORKDIR /app

# Copy files

COPY . .

COPY package.json /app/

# Add files (with extraction for archives)

ADD https://example.com/file.tar.gz /app/

# Run commands

RUN apt-get update && apt-get install -y python3

# Set environment variables

ENV NODE\_ENV=production

ENV PATH=$PATH:/app/bin

# Expose ports

EXPOSE 8080 3000

# Set user

USER nodejs

# Create volume mount point

VOLUME ["/data"]

# Set entrypoint

ENTRYPOINT ["python3"]

# Set default command

CMD ["app.py"]

# Add labels

LABEL version="1.0"

LABEL description="My application"

# Health check

HEALTHCHECK --interval=30s --timeout=3s --retries=3 \

CMD curl -f http://localhost:8080/ || exit 1

# Build arguments

ARG BUILD\_DATE

ARG VERSION=latest

**Multi-stage Dockerfile Example**

# Build stage

FROM node:16 AS builder

WORKDIR /app

COPY package\*.json ./

RUN npm install

COPY . .

RUN npm run build

# Production stage

FROM nginx:alpine

COPY --from=builder /app/dist /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

**Volumes**

**Volume Management**

# Create volume

docker volume create myvolume

# Create volume with driver options

docker volume create --driver local \

--opt type=nfs \

--opt o=addr=192.168.1.1,rw \

--opt device=:/path/to/dir \

myvolume

# List volumes with filters

docker volume ls --filter dangling=true

# Backup volume

docker run --rm -v myvolume:/data -v $(pwd):/backup \

alpine tar czf /backup/backup.tar.gz -C /data .

# Restore volume

docker run --rm -v myvolume:/data -v $(pwd):/backup \

alpine tar xzf /backup/backup.tar.gz -C /data

# Mount volume with specific options

docker run -v myvolume:/app:ro <image> # Read-only

docker run -v myvolume:/app:rw <image> # Read-write

# Anonymous volumes

docker run -v /app/data <image>

**Networks**

**Network Management**

# List networks

docker network ls

# Create network

docker network create mynetwork

# Create network with specific driver

docker network create --driver bridge mynetwork

docker network create --driver overlay mynetwork

# Create network with subnet

docker network create --subnet=172.20.0.0/16 mynetwork

# Connect container to network

docker network connect mynetwork <container\_id>

# Disconnect container from network

docker network disconnect mynetwork <container\_id>

# Run container on specific network

docker run --network mynetwork <image>

# Inspect network

docker network inspect mynetwork

# Remove network

docker network rm mynetwork

# Remove unused networks

docker network prune

# Create containers that can communicate

docker run -d --name web --network mynetwork nginx

docker run -d --name db --network mynetwork postgres

# Port publishing

docker run -p 127.0.0.1:8080:80 <image> # Bind to specific interface

docker run -P <image> # Publish all exposed ports

**Docker Compose**

**Basic Compose Commands**

# Start services

docker-compose up

docker-compose up -d # Detached mode

# Start specific service

docker-compose up <service\_name>

# Build services

docker-compose build

docker-compose build --no-cache

# Stop services

docker-compose down

docker-compose down --volumes # Remove volumes too

# View running services

docker-compose ps

# View logs

docker-compose logs

docker-compose logs <service\_name>

docker-compose logs -f # Follow logs

# Execute command in service

docker-compose exec <service\_name> <command>

# Scale services

docker-compose up --scale web=3

# Restart services

docker-compose restart <service\_name>

# Pull latest images

docker-compose pull

# Validate compose file

docker-compose config

# Run one-off command

docker-compose run <service\_name> <command>

**Sample docker-compose.yml**

version: '3.8'

services:

web:

build: .

ports:

- "8080:80"

environment:

- NODE\_ENV=production

volumes:

- ./src:/app/src

depends\_on:

- db

networks:

- app-network

db:

image: postgres:13

environment:

POSTGRES\_DB: mydb

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

volumes:

- db-data:/var/lib/postgresql/data

networks:

- app-network

volumes:

db-data:

networks:

app-network:

driver: bridge

**Orchestration**

**Container Orchestration Concepts**

# Initialize Docker Swarm

docker swarm init

# Join swarm as worker

docker swarm join --token <token> <manager-ip>:2377

# Deploy stack

docker stack deploy -c docker-compose.yml mystack

# List stacks

docker stack ls

# List stack services

docker stack services mystack

# Remove stack

docker stack rm mystack

# Update service

docker service update --image nginx:latest myservice

# Scale service

docker service scale myservice=5

# Rolling updates

docker service update --update-parallelism 2 --update-delay 10s myservice

**Docker Swarm**

**Swarm Management**

# Initialize swarm mode

docker swarm init --advertise-addr <manager-ip>

# Add labels to nodes

docker node update --label-add role=frontend <node-id>

# List nodes

docker node ls

# Inspect node

docker node inspect <node-id>

# Promote worker to manager

docker node promote <node-id>

# Demote manager to worker

docker node demote <node-id>

# Remove node from swarm

docker node rm <node-id>

# Drain node (stop scheduling tasks)

docker node update --availability drain <node-id>

# Make node active again

docker node update --availability active <node-id>

# Leave swarm

docker swarm leave --force # For managers

**Services**

**Service Management**

# Create service

docker service create --name myservice nginx

# Create service with replicas

docker service create --name myservice --replicas 3 nginx

# Create service with port publishing

docker service create --name myservice -p 8080:80 nginx

# Create service with constraints

docker service create --name myservice \

--constraint 'node.labels.role==frontend' nginx

# List services

docker service ls

# Inspect service

docker service inspect myservice

# View service logs

docker service logs myservice

# List service tasks

docker service ps myservice

# Update service image

docker service update --image nginx:alpine myservice

# Add environment variable

docker service update --env-add NODE\_ENV=production myservice

# Add mount

docker service update --mount-add type=volume,src=myvolume,dst=/data myservice

# Remove service

docker service rm myservice

# Force update (recreate containers)

docker service update --force myservice

**YAML Files**

**Docker Compose YAML Structure**

version: '3.8'

# Services definition

services:

# Web service

web:

# Build configuration

build:

context: ./web

dockerfile: Dockerfile.dev

args:

- NODE\_ENV=development

# Image specification

image: myapp:latest

# Container name

container\_name: web-container

# Port mapping

ports:

- "3000:3000"

- "443:443"

# Environment variables

environment:

- DATABASE\_URL=postgresql://user:pass@db:5432/mydb

- REDIS\_URL=redis://redis:6379

# Environment file

env\_file:

- .env.development

# Volume mounts

volumes:

- ./src:/app/src:ro

- node\_modules:/app/node\_modules

- logs:/app/logs

# Network configuration

networks:

- frontend

- backend

# Dependencies

depends\_on:

db:

condition: service\_healthy

# Health check

healthcheck:

test: ["CMD", "curl", "-f", "http://localhost:3000/health"]

interval: 30s

timeout: 10s

retries: 3

# Restart policy

restart: unless-stopped

# Resource limits

deploy:

resources:

limits:

cpus: '0.50'

memory: 512M

reservations:

cpus: '0.25'

memory: 256M

# Labels

labels:

- "traefik.enable=true"

- "traefik.http.routers.web.rule=Host(`example.com`)"

# Database service

db:

image: postgres:13

container\_name: postgres-db

environment:

POSTGRES\_DB: mydb

POSTGRES\_USER: user

POSTGRES\_PASSWORD: password

volumes:

- db\_data:/var/lib/postgresql/data

- ./init.sql:/docker-entrypoint-initdb.d/init.sql:ro

networks:

- backend

healthcheck:

test: ["CMD-SHELL", "pg\_isready -U user -d mydb"]

interval: 30s

timeout: 10s

retries: 5

# Redis service

redis:

image: redis:alpine

container\_name: redis-cache

volumes:

- redis\_data:/data

networks:

- backend

command: redis-server --appendonly yes

# Volume definitions

volumes:

db\_data:

driver: local

redis\_data:

driver: local

node\_modules:

logs:

# Network definitions

networks:

frontend:

driver: bridge

backend:

driver: bridge

internal: true

# Secrets (Docker Swarm)

secrets:

db\_password:

file: ./secrets/db\_password.txt

api\_key:

external: true

# Configs (Docker Swarm)

configs:

nginx\_config:

file: ./config/nginx.conf

**Docker Stack YAML Example**

version: '3.8'

services:

web:

image: nginx:alpine

ports:

- "80:80"

deploy:

replicas: 3

update\_config:

parallelism: 1

delay: 10s

restart\_policy:

condition: on-failure

delay: 5s

max\_attempts: 3

networks:

- webnet

configs:

- source: nginx\_config

target: /etc/nginx/nginx.conf

secrets:

- ssl\_cert

- ssl\_key

db:

image: postgres:13

environment:

POSTGRES\_DB: mydb

POSTGRES\_USER: user

POSTGRES\_PASSWORD\_FILE: /run/secrets/db\_password

volumes:

- db-data:/var/lib/postgresql/data

deploy:

replicas: 1

placement:

constraints:

- node.role == manager

networks:

- webnet

secrets:

- db\_password

volumes:

db-data:

networks:

webnet:

driver: overlay

configs:

nginx\_config:

external: true

secrets:

db\_password:

external: true

ssl\_cert:

external: true

ssl\_key:

external: true

**Additional Useful Commands**

**System Information**

# Show Docker version

docker version

# Show Docker system information

docker info

# Show disk usage

docker system df -v

# Monitor events

docker events

# Show Docker daemon logs

journalctl -u docker.service

# Update Docker

sudo apt-get update && sudo apt-get install docker-ce docker-ce-cli containerd.io

**Debugging & Troubleshooting**

# Attach to running container

docker attach <container\_id>

# Run container with debugging

docker run --rm -it --entrypoint /bin/bash <image>

# Check container resource usage

docker stats --no-stream

# Export container filesystem

docker export <container\_id> > container.tar

# Import container filesystem

docker import container.tar myimage:latest

# Show container processes

docker top <container\_id>

# Show container changes

docker diff <container\_id>

This comprehensive guide covers all the Docker commands and concepts you requested. Each section includes practical examples and explanations to help you understand and use Docker effectively in your development and deployment workflows.